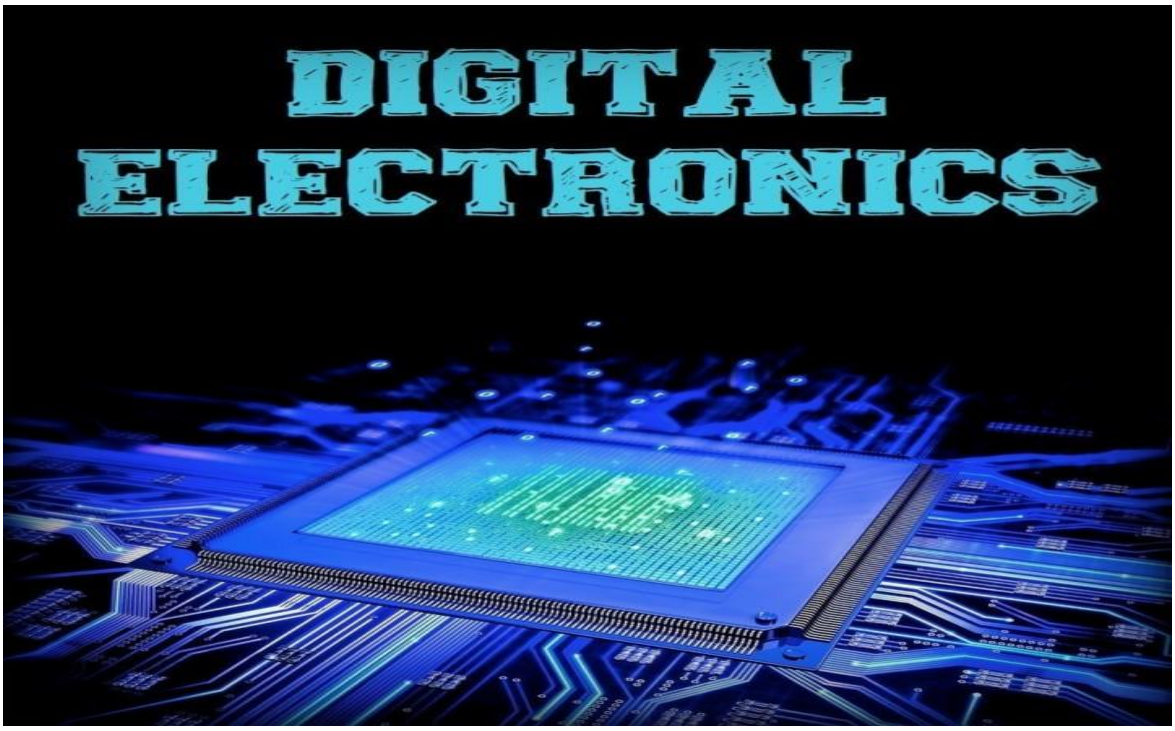
Abanob Evram

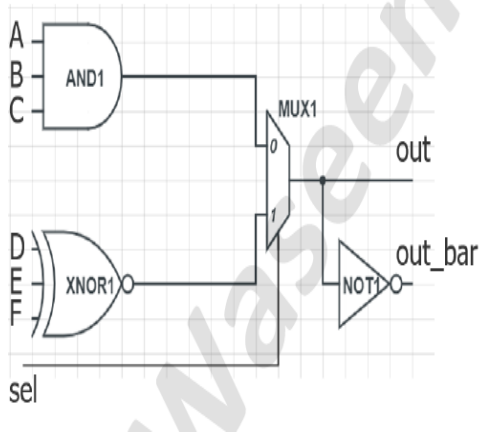
Assignmen1



[Q1]

The code:

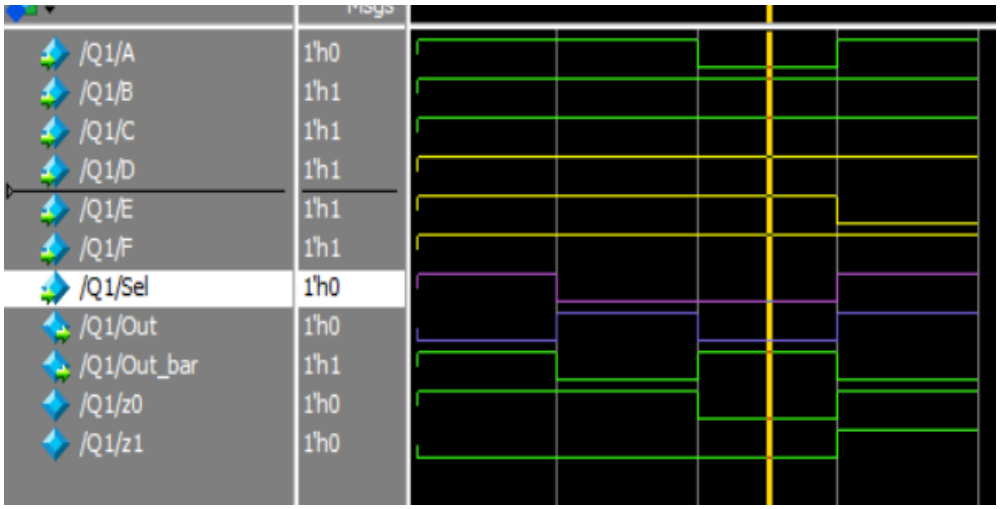
- The design has 7 inputs and 2 outputs
- Use assign statements to design the following




```
module Mux2(in0,in1,sel,out);
input in0,in1,sel;
output out;
assign out=(sel==1)?in1:in0;
endmodule
module Q1(A,B,C,D,E,F,Sel,Out,Out_bar);
input A,B,C,D,E,F,Sel;
output Out,Out_bar;
wire z0,z1;
assign z0 =A&B&C ;
assign z1=~(D^E^F);
Mux2 m1(.sel(Sel),.in0(z0),.in1(z1),.out(Out));
assign Out_bar=~(Out);
endmodule
```

Another Code :

```
module Q1(A,B,C,D,E,F,sel,out,outbar);
input A,B,C,D,E,F,sel;
output out,outbar;
wire z1,z0;
assign z0=A&B&C;
assign z1=~(D^E^F);
assign out=(sel==1)?z1:z0;
assign outbar=~(out);
endmodule
```



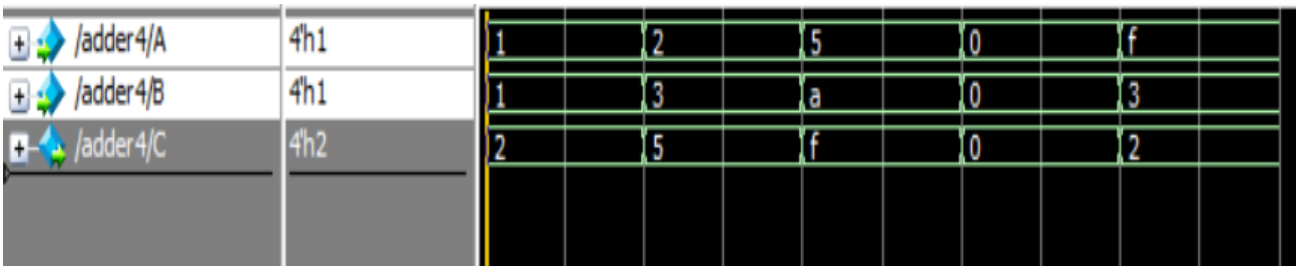
[Q2]

Implement 4-bit adder using addition operator

- The design takes 2 inputs (**A, B**) and the summation is assigned to output (**C**) ignoring the carry

The code:

```
module adder4(A,B,C);
input [3:0] A,B;
output [3:0] C;
assign C =A+B ;
endmodule
```



[Q3]

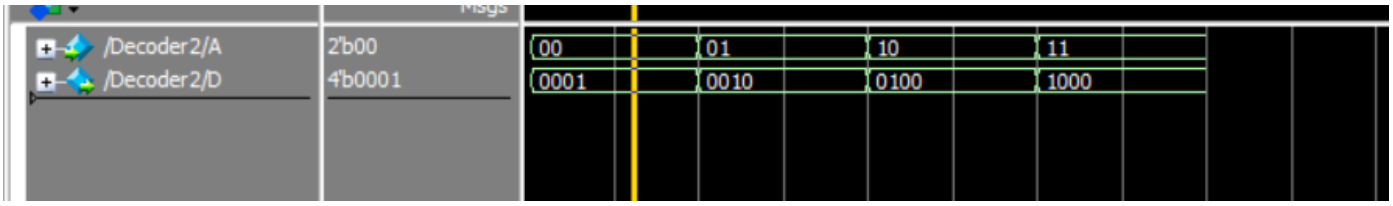
The code:

```
module Decoder2(A,D);
input [1:0] A ;
output reg [3:0] D;
always @(*) begin
        if (A==0)
        D='b0001;

        else if (A==1)
        D='b0010;
        else if (A==2)
       D='b0100;
        else if (A==3)
        D='b1000;
end
endmodule
```

3) Implement 2-to-4 Decoder using conditional operator (A logic decoder has n input lines and 2^n output lines. Each output line corresponds to a unique combination of the input values.)

- The design has input **A** (2 bits) and output **D** (4 bits)

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



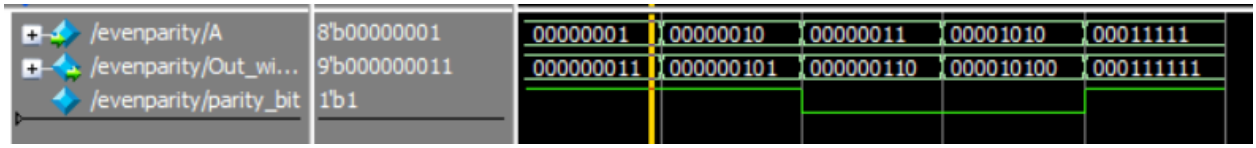
[Q4]

4) Implement an even parity generator module. In case you don't know what a parity bit is, please check this link. The design input is a bus where a reduction operator will be used to generate the even parity bit.

- The design has 1 input **A** (8 bits) and 1 output **out_with_parity** (9 bit) where the parity bit calculated will be inserted in the least significant bit of the output bus and the remaining bits will be the input A (Hint: use concatentation).

The code:

```
module evenparity(A,Out_with_parity);

input [7:0]A ;

output [8:0] Out_with_parity;

wire parity_bit;

assign parity_bit = ^A;

assign Out_with_parity ={A,parity_bit} ;

endmodule
```



[Q5]

The code:

```
module comparator2(A,B,greater,equal,less);
input [3:0] A,B;
output reg greater,equal,less;
always @(*) begin
        if (A>B) begin
    greater=1;
        equal=0;
        less=0;
        end
        else if (A==B) begin
        greater=0;
        equal=1;
        less=0;
        end
    else if(A<B)begin
    greater=0;
    equal=0;
    less=1;
    end
end
endmodule
```

Implement a comparator that compares 2 inputs (**A, B**) and has 3 outputs using conditional operator.

- The first output **A_greaterthan_B** is high only when A is greater than B
- The second output **A_equals_B** is high only when A equals B
- The third output **A_lessthan_B is** high only when A is less than B